1. What is CTE?

1. A Common Table Expression (CTE) is a temporary named result set that you can reference within a SELECT, INSERT, UPDATE, DELETE, or MERGE statement. It exists only during the execution of a single query and makes complex queries more readable and maintainable.

   Basic Syntax:

```
WITH CTE_Name [ (Column1, Column2, ...) ] AS
(
    -- Your SQL Query here
    SELECT ...)
-- Main query that uses the CTE
SELECT * FROM CTE_ame;

                    N
```

2. Why Use a CTE? (The Benefits)

2. **Improve Readability:** Break down a very long and complex query into smaller, logical chunks. Instead of having one giant query with many subqueries, you can name each part.
3. **Recursive Queries (T-SQL's Superpower):** Solve problems that involve parent-child relationships, which are impossible with a simple `SELECT`.
4. **Replace a View:** If you need a temporary view for just one query, a CTE is perfect. You avoid creating a permanent view in the database.
5. **Use Multiple Times:** You can reference the CTE multiple times in the main query. (In some cases, this might be more efficient than writing a subquery multiple times, though the database engine often optimizes them similarly).

3. Types of CTEs

1. Simple (Non-Recursive) CTE

Used for breaking down complex queries into simpler, logical parts.

```
WITH EmployeeCTE AS
(
  SELECT EmployeeID, Name, Salary, DepartmentID
  FROM Employees
  WHERE Salary > 50000
)
SELECT * FROM EmployeeCTE
WHERE DepartmentID = 5;
```

2. Recursive CTE

References itself to work to build a final answer step-by-step. or tree-structured data. Must have an anchor member and a recursive member.

```
WITH EmployeeHierarchy AS
(
  -- Anchor member
  SELECT EmployeeID, Name, ManagerID, 1 AS Level
  FROM Employees
  WHERE ManagerID IS NULL

  UNION ALL

  -- Recursive member
  SELECT e.EmployeeID, e.Name, e.ManagerID, eh.Level + 1
  FROM Employees e
  INNER JOIN EmployeeHierarchy eh ON e.ManagerID = eh.EmployeeID
)
SELECT * FROM EmployeeHierarchy;
```

3. Multiple CTEs

You can define multiple CTEs in a single query, separated by commas.

```
WITH
DeptCTE AS (
  SELECT DepartmentID, DepartmentName FROM Departments
),
EmpCTE AS (
  SELECT EmployeeID, Name, DepartmentID FROM Employees
)
SELECT e.Name, d.DepartmentName
FROM EmpCTE e
INNER JOIN DeptCTE d ON e.DepartmentID = d.DepartmentID;
```

Advantages

1. Improved Readability: Makes complex queries easier to understand by breaking them into logical, named sections
2. Code Reusability: You can reference the same CTE multiple times in a single query without repeating the code
3. Recursive Queries: Enables working with hierarchical data like organizational charts, bill of materials, or folder structures
4. Maintainability: Easier to debug and modify compared to nested subqueries
5. Alternative to Views: Useful when you need a temporary result set without creating a permanent view
6. Better Organization: Separates query logic into manageable chunks, especially useful for complex calculations

Disadvantages

1. Performance Limitations:

- CTEs are not indexed and don't store statistics
- The query optimizer may not always optimize CTEs efficiently
- Each reference to a CTE may cause it to be executed again (no caching)

2. Scope Limitation: CTEs exist only for a single query execution; they can't be reused across multiple queries
3. No Parameter Support: Unlike stored procedures or functions, CTEs cannot accept parameters
4. Recursive CTE Risks:

- Can cause infinite loops if not properly constrained
- May hit the default recursion limit (100 levels) requiring MAXRECURSION hint
- Can consume significant resources with large hierarchies

5. Memory Consumption: Large result sets in CTEs can consume considerable memory
6. Limited Optimization: Unlike temporary tables, you cannot create indexes or statistics on CTEs
7. Debugging Challenges: Recursive CTEs can be difficult to troubleshoot when they don't produce expected results

4. Real-World Scenarios & Examples

Scenario 1: Improving Readability (Standard SQL & T-SQL)

Problem: You want a report showing all employees and the name of their department. The data is in two tables: `Employees` and `Departments`.

The Messy Way (using subqueries):

```
SELECT
    e.EmployeeName,
    (SELECT d.DepartmentName FROM Departments d WHERE d.DepartmentID = e.Depart
mentID) as DeptName
FROM Employees e;
```

The Clean Way (using a CTE):

```
-- Step 1: Create a CTE that gets DepartmentNames
WITH DepartmentCTE AS (
    SELECT DepartmentID, DepartmentName
    FROM Departments
)
-- Step 2: Use the CTE in the main query
SELECT e.EmployeeName, d.DepartmentName
FROM Employees e
INNER JOIN DepartmentCTE d ON e.DepartmentID = d.DepartmentID;
```

Why it's better: The logic for getting department data is separated and named, making the main `JOIN` logic much clearer.

5. Alternative Options to CTEs

1. **Subqueries:**
   - **What it is:** A query nested inside another query (`SELECT ... FROM (SELECT ...)`).
   - **When to use it:** For very simple, one-off calculations or filters.
   - **CTE vs. Subquery:** CTEs are almost always more readable for complex logic.
2. **Temporary Tables (`#TempTable`):**

- **What it is:** A physical table created in the `tempdb` database. It exists for the entire session/connection, not just one query.
- **When to use it:**

- You need to use the result set in **multiple, separate queries**.
- The intermediate result is very large, and you want to build indexes on it for performance.

- **CTE vs. Temp Table:** A CTE is generally faster to write for a single, complex query. A Temp Table is more powerful for complex, multi-step data processing.

3. **Views:**

- **What it is:** A saved, permanent `SELECT` query in the database.
- **When to use it:** You have a specific data perspective that you need to use over and over again across many different applications and queries.
- **CTE vs. View:** Use a CTE for a one-time need. Use a View for a reusable business logic.

6. Summary Table: When to Use What

Feature	Purpose	Scope/Lifetime	Best For
CTE	Organizing a single complex query; Recursion.	Single Query Execution	Readability, recursive problems, replacing a view for one query.
Subquery	Simple nested calculations or filters.	Single Query Execution	Quick, simple tasks where a CTE would be overkill.
Temp Table	Storing intermediate results for complex processing.	Current Session/Connection	Breaking down very complex processes, sharing data between multiple queries, performance tuning with indexes.
View	Saving a query definition for reuse.	Permanent Database Object	Reusable business logic that many users/applications need.